Jarosław Milewski

```
PPPPP     AA     SSSS    CCCC     AA    LL
PP  PP   AAAA   SS  SS  CC  CC   AAAA   LL
PP  PP  AA  AA  SS      CC      AA  AA  LL
PPPPP   AAAAAA   SSSS   CC      AAAAAA  LL
PP      AA  AA      SS  CC  CC  AA  AA  LL
PP      AA  AA  SSSSS    CCCC   AA  AA  LLLLLL
```

```
     444   0000    0000
    4444  00  00  00  00
   44 44  00  00  00  00
  44  44  00  00  00  00
  444444  00  00  00  00
      44   0000    0000
```

wersja 2.1 dla systemu CROOK-5

P O D R Ę C Z N I K U Ż Y T K O W N I K A

Warszawa, lipiec 1987

Wstęp

Kompilator Pascal 400 wersja 2.1 jest przeznaczony dla minikomputerów MERA-400 i MX-16 działających pod systemem operacyjnym CROOK-5. Niniejszy podręcznik zawiera opis różnic Pascala 400 w stosunku do Pascala wzorcowego oraz zasady kompilowania i wykonywania programów. Pełny opis języka Pascal znaleźć można w natępujących publikacjach :

[1] Specification for computer programming language Pascal, British Standards Institution BS-6192, 1982.

[2] K. Jensen, N. Wirth : Pascal user manual and report, Springer-Verlag, New York 1976.

[3] M. Iglewski, J. Madey, S. Matwin : Pascal - Język wzorcowy, Pascal 6000, WNT Warszawa 1979.

Użytkownik Pascala 400 przygotowuje zwykle programy do wykonania za pomocą makrozlecenia kompilacyjnego, opisanego w p. 5.2. Makrozlecenie obejmuje dwa przebiegi kompilatora, asemblację i konsolidację kodu. Osoby pragnące posiadać pełniejszą kontrolę nad procesem kompilacji, a także chcące korzystać z nakładkowania i własnych bibliotek podprogramów, powinny zapoznać się dodatkowo z podręcznikami programów GASS (asembler), LINK (konsolidator) i LIAR (bibliotekarz):

[4] P. Findeisen, P. Gburzyński : GASS - Mera 400 Assembly Language Reference Manual (Part One & Two), Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego nr 117 i 118, Warszawa, 1982.

[5] J. Rudziński : Podręcznik użytkownika asembera GASS w systemie operacyjnym CROOK, IIUW Warszawa, październik 1986.

[6] J. Milewski, J. Rudziński : Podręcznik użytkownika programu redagująco-łączącego LINK dla systemu CROOK, IIUW Warszawa, marzec 1986.

[7] J. Milewski, J. Rudziński : Podręcznik użytkownika bibliotekarza LIAR dla systemu CROOK, IIUW Warszawa, marzec 1986.

Jako podręcznik samego systemu CROOK można polecić

[8] Praca zbiorowa pod redakcją W.J. Martina : System operacyjny CROOK-4 dla minikomputera MERA-400 (wersja 58/19), Kursy Zastosowań Matematyki, Instytut Matematyczny PAN, Warszawa, 1986 (z późniejszymi zmianami dotyczącymi systemu CROOK-5).

Pascal 400 oparty jest na tym samym przenaszalnym kompilatorze, który stał się w przeszłości podstawą opracowania znanych systemów Pascal 360 i Pascal 6000. Efektem wspólnego pochodzenia jest zbliżona charakterystyka funkcjonalna, m.in. niemal identyczna diagnostyka błędów. Kod wynikowy uzyskiwany z Pascala 400 odznacza się stosunkowo dużą zwartością i szybkością: nienekładkowane programy rzędu 5000 linii tekstu źródłowego mieszczą się w obszarze poniżej 32K pamięci. Pełne przetworzenie programu tej wielkości za pomocą makrozlecenia PASCAL trwa na nieobciążonej maszynie typu MERA-400 około 15 minut.

0. Leksyka.

Linia programu źródłowego nie może zawierać więcej niż 128 znaków. We wszystkich nazwach istotne jest tylko pierwszych 8 znaków. Kompilator akceptuje zarówno małe jak i duże litery, jednak nazwy różniące się tylko pocztem liter są nierozróżnialne.

Nawiasy kwadratowe można zastępować symbolami (. i .), a nawiasy klamrowe symbolami (* i *). Zamiast strzałki w górę stosowany jest znak @.

1. Stałe.

W stałych napisowych można umieszczać dowolne znaki kodu ISO-7 (ASCII). Długość napisu nie może przekraczać 80 znaków. Nie akceptowane są definicje stałych typu SET postaci [1..5] lub ['B'..'X'] (ukryta definicja typu okrojonego). Żądaną stałą można otrzymać wyliczając wszystkie jej elementy, np. [1,2,3,4,5].

2. Typy.

2.1. Typy proste.

INTEGER - zakres -32768 .. 32767;
        - rozmiar 1 słowo maszynowe.

REAL   - zakres bezwzględny 1.47E-39 .. 1.70E38;

- dokładność względna 3.73E-9;
- rozmiar 3 słowa maszynowe,
  w tym mantysa 40 bitów, cecha 8 bitów.

CHAR - pełny kod ASCII;
- rozmiar 1 słowo maszynowe;

BOOLEAN - reprezentacja FALSE = 0; TRUE = 1;
- rozmiar 1 słowo maszynowe.

2.2. Typy strukturalne.

SET - Zbiór może zawierać do 128 elementów. Każda zmienna typu SET zajmuje 8 słów maszynowych.

RECORD - W rekordach z wariantami musi występować pole znacznikowe.

ARRAY - Tablice znakowe można wypisywać za pomocą procedur standardowych WRITE/WRITELN oraz porownywać leksykograficznie używając symboli relacyjnych <, <=, =, <>, >=, >.

FILE - Nie jest dozwolone deklarowanie plików jako składowych typów strukturalnych. Deklaracje plików mogą występować zarówno w programie głównym jak i w podprogramach, przy czym deklaracja plików w nagłówku programu jest pusta semantycznie. W miejsce standardowych plików INPUT i OUTPUT wprowadzone zostały pliki o nazwach odpowiednio SI i SO.

Każdy plik jest automatycznie otwierany przy wejściu do jednostki programowej, w której został zadeklarowany i automatycznie zamykany przy jej zakończeniu.

Uwaga! W obecnej wersji kompilatora zmienna buforowa pliku, alokowana na stosie roboczym programu, musi znaleźć się w bajtowym obszarze przestrzeni adresowej (dolne 32K). Narzuca to z jednej strony ograniczenie na wielkość kodu programu, a z drugiej może utrudniać deklarowanie plików w zbyt głęboko zagnieżdżonych podprogramach.

Na poziomie job-control'a pliki reprezentowane są przez strumienie o nazwach składających się z pierwszych trzech liter nazwy pliku, np. PRINT -> PRI. Dotyczy to w szczególności standardowych plików SI i SO, którym odpowiadają strumienie o tych samych nazwach. Należy pamiętać o przywiązaniu wszystkich strumieni (plików) do odpowiednich zbiorów przed wywołaniem programu, lub o skorzystaniu z

bibliotecznych procedur ASSIGN lub NEXTFID (por. 4.3).

Plików SI i SO nie należy deklarować w programie. Zamknięcie pliku tekstowego nie powoduje automatycznego dopisania znaku konca linii za automatycznego dopisania znaku końca linii za ostatnią linią pliku, ani opróżnienia bufora (ostatnią operacją pisania na plik tekstowy powinno więc być zawsze WRITELN).

Do operowania plikami służą procedury standardowe READ, READLN, WRITE, WRITELN, RESET i REWRITE oraz funkcje EOF i EOLN. Niedozwolone jest natomiast bezposrednie posługiwanie się zmienną buforową pliku (np. SI@) oraz korzystanie z procedur GET i PUT, które nie są obiektami standardowymi Pascala 400. Szczegółowy opis działania procedur we/wy znaleźć można w punkcie 4.2.

W Pascalu 400 nie są dostępne typy upakowane. Kompilator akceptuje wprawdzie deklaracje PACKED <typ> lecz są one traktowane jak deklaracje bez słowa kluczowego PACKED.

3. Instrukcje.

Dopuszczalne są wszystkie instrukcje Pascala wzorcowego. Jedyną zmianą jest rozszerzenie instrukcji CASE o mozliwość wystąpienia selektora domyślnego postaci "ELSE <instrukcja>" po ostatnim elemencie listy wyboru np.

```
CASE CH OF
  'A','B' : I:=1;
  'X','Y' : I:=2
       ELSE I:=0
END;
```

4. Jednostki programowe

4.1. Jednostki programowe deklarowane przez użytkownika.

Podprogramy i program główny kompilowane są na niezależne moduły binarne. Nazwy modułów są zgodne z nazwami odpowiednich jednostek w programie źródłowym, przy czym muszą być one różne w obrębie całego programu z dokładnością do pierwszych 8

znaków. Ograniczenia te wynikają z cech konsolidatora LINK.

Ze względu na zastosowanie prostego display'a parametry proceduralne i funkcyjne wykonywane są w środowisku statycznym, tzn. takim, jakim jest ono w momencie wywołania a nie przekazania ich jako parametrów.

Deklaracje parametrów proceduralnych/funkcyjnych w nagłówku podprogramu określają jedynie ich nazwy i ewentualne typy, ale nie ich własne parametry, np.

```
FUNCTION CALKA(A,B,EPS: REAL;
               FUNCTION F: REAL): REAL;
```

ale nie

```
FUNCTION CALKA(A,B,EPS: REAL;
               FUNCTION F(X,Y: REAL): REAL): REAL;
```

Cecha ta jest zgodna z oryginalnym raportem języka [2], natomiast niezgodna z późniejszą normą [1]. Jedną z jej konsekwencji jest to, iż parametry proceduralne/funkcyjne nie mogą same posiadać parametrów proceduralnych/funkcyjnych.

4.2. Podprogramy standardowe.

Standardowe funkcje numeryczne mogą być aktualnymi parametrami funkcyjnymi. GET, PUT, PAGE, PACK i UNPACK nie sa obiektami standardowymi języka (PAGE jest dostępna jako procedura biblioteczna), wprowadzono natomiast dwie dodatkowe procedury standardowe MARK i RELEASE do operacji na zmiennych wskazywanych ("dynamicznych"). Ponizej opisane zostały podprogramy standardowe Pascala 400, w ktorych wystepują zmiany w stosunku do języka wzorcowego, lub ograniczenia implementacyjne.

MARK - Markuje wierzchołek heapu, czyli obszaru pamięci zawierającego zmienne wskazywane. Służy do współpracy z procedurą RELEASE.

RELEASE - Powoduje zwolnienie tego obszaru heapu, który został zaalokowany od ostatniego wywołania procedury MARK. Wartosci zmiennych wskazujących na zwolniony obszar są nieokreślone. Operacja markowania działa na zasadzie stosu, tzn. można wielokrotnie zamarkować heap procedurą MARK a nastepnie wielokrotnie użyć procedury RELEASE do zwolnienia zamarkowanych obszarów. Liczba operacji RELEASE nie może oczywiście przekroczyć liczby wykonanych operacji MARK, a maksymalna wysokość stosu wynosi 40

markowań.

DISPOSE - Procedurę DISPOSE można używać w jednym programie maksymalnie dla 20 różnych rozmiarów zmiennych wskazywanych. Uwaga! Nie należy stosować jednocześnie procedury DISPOSE oraz procedur MARK/RELEASE.

EOF - Warunek końca pliku jest spełniony wtedy, gdy wczytany został ostatni element pliku. Dla pliku tekstowego pod pojęciem ostatniego elementu należy rozumieć znak końca ostatniej jego linii.

EOLN - Warunek końca linii jest spełniony wtedy, gdy wczytany został ostatni znak przed końcem linii. W sumie więc typowy schemat wczytywania pliku tekstowego (powiedzmy SI) przedstawia się następująco:

```
WHILE NOT EOF DO
  BEGIN WHILE NOT EOLN DO
      BEGIN READ(ZNAK);
        PRZETWÓRZ(ZNAK)
      END;
    READLN
  END;
```

READ - W przypadku pliku nietekstowego wczytywanych jest dokładnie tyle słów, ile wynosi sumaryczny rozmiar zmiennych będących parametrami procedury. Wczytywana informacja nie jest interpretowana w żaden dodatkowy sposób.

Dalsza część opisu procedury READ(LN) stosuje się do plików tekstowych. Jeżeli parametr procedury jest zmienną typu INTEGER lub REAL, to wczytywana jest kolejna poprawnie sformatowana liczba. Ciągi znaków nie będące poprawnie sformatowanymi liczbami, w tym dowolne separatory, są pomijane. W przypadku gdy wczytywana liczba wykracza poza zakres typu zmiennej zwracana jest odpowiednio największa lub najmniejsza wartość danego typu. Przy niedomiarze zmiennopozycyjnym zwracana jest wartość 0.0 (por. 4.3, funkcja OVF).

Znak końca linii ('cr') wczytywany jest przez procedurę READ jako spacja (wtedy, gdy nie korzystamy z READLN). Procedura READ zakłada ponadto, że pierwszym znakiem linii jest znak sterujący (np. 'lf'), w związku z czym pierwszy znak jest zawsze ignorowany.

Jeśli pierwszy parametr READ/READLN nie jest plikiem, to czytany będzie domyślnie plik SI.

WRITE - W przypadku pliku nietekstowego wypisywanych jest dokładnie tyle słów, ile wynosi sumaryczny rozmiar wartości będących parametrami procedury. Wypisywana informacja nie jest interpretowana w żaden dodatkowy sposób.

Dalsza część opisu procedury WRITE(LN) stosuje się do plików tekstowych. Sposób wypisywania wartości parametru zależy od jego typu. Liczby typu INTEGER można wypisywać w postaci heksadecymalnej używając specjalnego formatu postaci <liczba>: <rozmiar pola>: 'X'. Liczby typu REAL wypisywane są w formacie z wykładnikiem. W celu wypisania liczby bez wykładnika należy stosować format <liczba>: <rozmiar pola>: <liczba cyfr po kropce>.

Domyślne rozmiary pól dla poszczególnych typów są następujące : CHAR - 1, INTEGER - 10, REAL - 20, BOOLEAN - 10. Maksymalny rozmiar deklarowanego pola może wynosić 150. W przypadku gdy liczba będąca parametrem procedury nie mieści się w zadeklarowanym polu, zamiast niej wypisany zostaje ciąg gwiazdek. Sekwencja <znak>: <rozmiar pola> powoduje powtórzenie znaku podaną liczbę razy.

Procedura WRITELN nie tylko dopisuje znak końca linii, lecz faktycznie wypisuje ją całą opróżniając bufor pliku. Przy okazji linia zaopatrywana jest w znak sterujący, którym standardowo jest 'lf' (patrz też PAGE i OVERPRINT, p. 4.3).

Jeśli pierwszy parametr WRITE/WRITELN nie jest plikiem, to zapisywany będzie domyślnie plik SO.

4.3. Podprogramy zewnętrzne i biblioteczne.

W Pascalu 400 istnieje możliwość deklarowania podprogramów zewnętrznych, które dołączane są do programu dopiero w trakcie jego konsolidacji. Wersje binarne podprogramów zewnętrznych znajdują się albo w standardowej bibliotece Pascala 400 (PASLIB), albo w innych zbiorach binarnych dostarczanych przez użytkownika.

Procedurę lub funkcję zewnetrzną deklaruje się poprzez umieszczenie słowa kluczowego EXTERNAL zamiast ciała danego podprogramu. Oznacza to, że podprogram zewnętrzny może posiadać oprócz nagłówka również deklaracje własnych, wewnętrznych procedur i funkcji. W praktyce słowo EXTERNAL

podawane jest najczęściej bezpośrednio po nagłówku podprogramu.

Podprogramy zewnętrzne przechowywane w standardowej bibliotece Pascala 400 mogą być w naturalny sposób traktowane jako przedłużenie procedur i funkcji standardowych włączonych bezpośrednio do języka. Jedyna różnica polega na tym, że muszą być one deklarowane w programie źródłowym, zgodnie z podanymi niżej schematami.

```
PROCEDURE DATE(VAR YY,MM,DD: INTEGER); EXTERNAL;
```

Parametrom procedury nadana zostaje aktualna data.

```
PROCEDURE TIME(VAR HH,MM,SS: INTEGER); EXTERNAL;
```

Parametrom procedury nadany zostaje bieżący czas.

```
FUNCTION OVF(VAR F: TEXT): BOOLEAN; EXTERNAL;
```

Przyjmuje wartość TRUE jeśli w wyniku ostatniej operacji czytania liczby z pliku F wystąpił nadmiar lub niedomiar.

```
PROCEDURE PAGE(VAR F: TEXT); EXTERNAL;
```

Przejście do nowej strony na wskazanym pliku. Mówiąc dokładniej, wywołanie PAGE spowoduje, że najbliższa instrukcja WRITELN dotycząca tego samego pliku dołączy 'ff' zamiast 'lf' na początku wypisywanej linii.

```
PROCEDURE OVERPRINT(VAR F: TEXT); EXTERNAL;
```

Nadrukowanie bieżącej linii na ostatnio wypisywanej. Mówiąc dokładniej, wywołanie OVERPRINT spowoduje, że najbliższa instrukcja WRITELN dotycząca tego samego pliku pozbawi linię jakiegokolwiek znaku sterującego (brak 'lf' lub 'ff').

```
PROCEDURE PROMPT(VAR F: TEXT; CH: CHAR); EXTERNAL;
```

Procedura zmienia znak zaproszenia (prompt) wysyłany na monitor użytkownika przy pracy interakcyjnej. Standardowym promptem jest ':'. Nie można zmienić promptu PMD, którym jest '*'.

```
TYPE PARRNG = -1..6;
     PAR = RECORD
             CASE PARTYP: PARRNG OF
                   -1,0 : ();
                 1,2,5 : (SHORT: INTEGER);
```

```
                    3 : (LONG: ARRAY [1..2] OF INTEGER);
                    4 : (FID: ARRAY [1..5] OF INTEGER);
                    6 : (STGLEN: INTEGER;
                         STG: ARRAY [1..80] OF CHAR)
            END;
PROCEDURE NEXTPAR(VAR P: PAR); EXTERNAL;
```

Każde wywwołanie procedury NEXTPAR podstawia na zmienną P kolejny parametr wywołania programu, przekazywany zgodnie z konwencją 064. Bliższego opisu konwencji 020/060/064 należy szukać w podręczniku CROOKa; podana przez nas definicja typu PAR stanowi uproszczony zapis tej konwencji w kategoriach Pascala i może być odpowiednio zmodyfikowana dla potrzeb programu. W szczególności, ograniczenie długości parametru napisowego do 80 znaków jest całkowicie arbitralne. Dalsza interpretacja wartości parametru jest wyłączną sprawą użytkownika.

```
TYPE FID = ARRAY [1..5] OF INTEGER;
PROCEDURE ASSIGN(VAR F: .. ; FD: FID); EXTERNAL;
```

Procedura wiąże plik F ze zbiorem o tytule FID; tytuł ten powinien być podany w odpowiednim formacie CROOKowskim (5 słów w kodzie R40, bliższy opis w podręcznikach systemu). Jeżeli zbiór o podanym tytule nie istnieje, to zostanie on utworzony.

Zwracamy uwagę na fakt, iż koncepcja języka Pascal narzuca konieczność zadeklarowania procedury ASSIGN z konkretnym typem pliku F, choć dla samej procedury bibliotecznej typ ten jest obojętny. Jeśli chcemy zastosować procedurę ASSIGN w jednym programie dla różnych typów plikowych, to należy zadeklarować ją wielokrotnie w różnych jednostkach programowych (być może stworzonych tylko na tę okazję). Kompilator Pascala zostanie wówczas "oszukany" w tym sensie, że będą to dla niego różne lokalne procedury ASSIGN, które dopiero w poźniejszej fazie konsolidacji zostaną skojarzone z tą samą procedurą biblioteczną.

Przykład. Chcemy zastosować procedurę ASSIGN dla pliku typu TEXT i pliku typu FILE OF INTEGER. Deklarujemy

```
TYPE BINARY = FILE OF INTEGER;

PROCEDURE ASSIGN1(VAR F: TEXT; FD: FID);
PROCEDURE ASSIGN(VAR F: TEXT; FD: FID); EXTERNAL;
BEGIN ASSIGN(F,FD) END;

PROCEDURE ASSIGN2(VAR F: BINARY; FD: FID);
PROCEDURE ASSIGN(VAR F: BINARY; FD: FID); EXTERNAL;
BEGIN ASSIGN(F,FD) END;
```

i następnie stosujemy procedurę ASSIGN1 do plików tekstowych i ASSIGN2 do plików binarnych.

```
PROCEDURE NEXTFID(VAR F: .. ); EXTERNAL;
```

Podstawowym zastosowaniem parametrów przekazywanych zgodnie z konwencją 064, a będących tytułami zbiorów, jest wiązanie ich z odpowiednimi zmiennymi plikowymi. Procedura NEXTFID obsługuje tę typową sytuację, łącząc w sobie działanie NEXTPAR i ASSIGN: pobiera ona kolejny parametr wywołania programu, interpretuje go jako pełny lub skrócony tytuł zbioru (PARTYP = 2,3,4) i wiąże go z plikiem F. Napotkanie przez NEXTFID parametru nieodpowiedniego rodzaju powoduje błąd.

Uwagi na temat stosowania NEXTFID do różnych typów plików takie same jak w procedurze ASSIGN.

```
PROCEDURE OPTION(I: INTEGER): BOOLEAN; EXTERNAL;
```

W celu skorzystania z funkcji OPTION należy nadać zbiorowi zawierającemu program atrybut 054. Możliwe jest wówczas przekazanie programowi jednego parametru numerycznego, którego i-ty bit, licząc od prawej (i = 1..16), jest zwracany jako boolowska wartość funkcji OPTION.

```
PROCEDURE HALT; EXTERNAL;
```

Natychmiastowe przerwanie wykonywania programu ze statusem błędu 12. Jeśli w trakcie kompilacji programu była zapalona odpowiednia opcja, to błąd ten zostanie przechwycony przez interakcyjny debugger PMD (por. 6.1). W związku z tym procedura HALT może być również używana do dynamicznego śledzenia wykonania programu; uwadze Czytelnika polecamy też

```
PROCEDURE ASSERT(B: BOOLEAN);
BEGIN IF NOT B THEN HALT END;
```

która posłuży do dynamicznego sprawdzania asercji semantycznych programu.

```
TYPE CH8 = ARRAY [1..8] OF CHAR;
PROCEDURE OVERLAY(NAME: CH8); EXTERNAL;
```

Nakładkowanie stanowi najprostszy chyba, ale i najmniej wygodny sposób redukowania wymagań pamięciowych programów. Nie jest to z pewnością narzędzie, jakie można by polecać początkującemu programiście: stosowanie nakładek wymaga bardzo dokładnego zrozumienia mechanizmów konsolidacji i ładowania.

Środki projektowania i budowania struktury nakładek są dostarczane przez konsolidator LINK, i w jego też podręczniku należy szukać odpowiednich informacji. Pascal jako taki oferuje jedynie procedurę biblioteczną OVERLAY, służącą do ładowania nakładek o wskazanych nazwach.

Z samej natury języka Pascal wynika, że nakładkowany może

być tu jedynie kod programu, a nie jego zmienne (te ostatnie przechowywane są na stosie i heapie, i rządzą się własnymi zasadami alokacji i dealokacji). Ponadto, fakt kompilowania poszczególnych jednostek programowych Pascala na oddzielne moduły binarne sugeruje, że przeładowanie kodu może nastąpić tylko jednocześnie z wywołaniem lub powrotem z podprogramu. W związku z tym, działanie procedury

```
OVERLAY('NAZWA   ')
```

polega na pozostawieniu w running-systemie informacji, na podstawie której najbliższe wywołanie lub powrót z podprogramu będzie połączone z załadowaniem nakładki o wskazanej nazwie. Mechanizm ten działa poprawnie bez względu na to, czy nakładka "przykrywa" bieżące miejsce sterowania.

W praktyce wywołanie procedury OVERLAY należy umieszczać bezpośrednio przed wywołaniem podprogramu leżącego na innej nakładce, lub bezpośrednio przed słowem END kończącym ciało podprogramu. Warto przy tym mieć na uwadze następujące dodatkowe okoliczności:

(1) Wszystkie procedury running-systemu Pascala dobrze jest umieszczać w nakładce poziomu 0, niemniej ostateczne decyzje na temat struktury nakładek opierają się wyłącznie na mapie wzajemnych odwołań jednostek programowych dostarczanej przez LINK.

(2) Standardowe procedury wejścia/wyjścia i funkcje numeryczne również są "czułe" na informację pozostawioną przez OVERLAY.

(3) W zagnieżdżonych wyrażeniach w pierwszej kolejności wykonywane są pierwsze od lewej najgłębiej zagnieżdżone nazewniki funkcyjne. Załóżmy dla przykładu istnienie nakładki MAD, procedury SAD i funkcji BAD. Sekwencja

```
OVERLAY('MAD     '); SAD(BAD(X));
```

spowoduje załadowanie nakładki MAD w chwili wywołania funkcji BAD, a nie procedury SAD.

5. Kompilacja programów.

5.1. Opcje i strumienie kompilatora.

Pascal 400 jest kompilatorem dwuprzebiegowym (analiza syntaktyczna i generacja kodu pośredniego, optymalizacja i generacja kodu wynikowego), którym to przebiegom odpowiadają dwa oddzielne programy o nazwach PASCOM i PASGEN. Programy te

komunikują się przez zbiór roboczy (scratch) zawierający kod pośredni - przemieszany z (też "pośrednim") listingiem programu źródłowego. W sumie przebieg PASCOM korzysta ze strumieni

SI - tekst źródłowy,
SO - scratch,

a przebieg PASGEN ze strumieni

SI - scratch,
SO - kod wynikowy,
LO - listing.

Jeśli chodzi o opcje kompilatora, to zrezygnowano z $-komentarzy w tekście programu źródłowego, wprowadzając zamiast tego następujace opcje podawane przy wywołaniu kompilatora i obowiązujące dla całego programu:

1 - drukowanie listingu,
2 - generacja kodu wynikowego,
3 - kontrola zakresu wartości w instrukcjach wyboru, odwołaniach do tablic i podstawieniach na zmienne skalarne,
4 - wykonanie programu z możliwością użycia PMD po błędzie.

W obecnej wersji kompilatora opcje te należy skombinować w postaci pojedynczego parametru całkowitego, którego kolejne bity (licząc od prawej) oznaczają kolejne opcje. Dla przykładu,

```
PASCOM 11
PASGEN 11
```

(11 dziesiętnie = 1011 binarnie) oznacza kompilację programu z listingiem, generacją kodu i wykorzystaniem PMD, ale bez dodatkowej kontroli wybranych instrukcji. Wartość parametru w obydwu przebiegach powinna być taka sama.

Zwracamy uwage na fakt, ze w celu przygotowania programu do
_ - otowania programu do wykonania potrzebne są jeszcze dwa dodatkowe przebiegi: kod wynikowy uzyskany z PASGEN jest asemblowany przez GASS, a otrzymany zbiór modułów relokowalnych jest następnie konsolidowany przez LINK. W sumie więc przygotowanie programu do wykonania może wyglądać na przykład tak:

```
ASS SI SOURCE
ASS SO PCODE 0
PASCOM 7
ASS SI PCODE
ASS SO GCODE 0
ASS LO LP2
PASGEN 7
ASS GI GCODE
```

```
ASS GO LCODE 0
GASSB 4
LINK LCODE ?LIB PASLIB ?OUT PGM
ERF PCODE GCODE LCODE
SET PGM ATR 064 (względnie 054)
```

Sekwencję tę wygodnie jest sparametryzować i zamknąć w formie makrozlecenia job-controla. Standardowe makrozlecenie dostarczane wraz z Pascalem 400 opisane jest w następnym punkcie.

5.2. Makrozlecenie PASCAL

Makrozlecenie PASCAL organizuje czynności opisane w 5.1, a jego wywołanie ma w ogólności postać

```
PASCAL źródło wynik [listing] [opcje]
```

gdzie

źródło - tytuł zbioru zawierającego tekst źródłowy programu,

wynik - tytuł zbioru, na na którym zapisany zostanie kod wynikowy,

listing - tytuł zbioru przeznaczonego dla listingu programu (domyślnie MON),

opcje - mogą być podane w dowolnej kolejności, a ich znaczenie jest wyjaśnione poniżej.

Standardowo wykonywane są wszystkie 3 fazy obróbki programu: kompilacja (2 przebiegi), asemblacja i konsolidacja. Zbiór wynikowy zawiera wówczas kod gotowy do wykonania, to znaczy pojedynczy, absolutny moduł binarny obejmujący zarówno cały program jak i wszystkie dołączone do niego procedury runtime'owe i zewnętrzne. Stosując opcję ?COM lub ?ASM możemy poprzestać na którejś z faz wcześniejszych:

?ASM - sama kompilacja. Zbiór wynikowy zawiera kod programu w postaci tekstu asemblera GASS.

?COM - kompilacja i asemblacja, bez konsolidacji. Zbiór wynikowy zawiera relokowalne moduły binarne odpowiadające poszczególnym jednostkom programowym (procedury, funkcje, program główny). Zbiór taki może być źródłem dla konsolidatora LINK lub bibliotekarza LIAR.

Następna grupa opcji powiela w nieco wygodniejszej formie opcje samego kompilatora:

?PMD - włączenie debuggera PMD (domyślnie wyłączony),

?NOC - wyłączenie kontroli skalarów i tablic (domyślnie włączona),

?FUL - listing całego programu (domyślnie tylko linie z błędami).

I wreszcie ostatnia grupa związana jest z fazą konsolidacji:

?MAP - (pełna) mapa ładowania programu zostanie dopisana do pliku listowania (pomocne przy projektowaniu struktury nakładek),

?LIB lib1 .. libn - deklaracja dodatkowych bibliotek użytkownika, które mają być przeglądane przed użytkownika, które mają być przeglądane przed standardową biblioteką Pascala,

?OBJ fil1 .. filn - deklaracja dodatkowych zbiorów zawierających binarne moduły relokowalne, które mają być dołączone do programu jeszcze przed przeglądaniem bibliotek (w praktyce rzadko stosowane).

W przypadku nieprawidłowego użycia makrozlecenia PASCAL podaje ono zwięzłą informację na temat dostępnych opcji.

5.3. Listing programu.

Każda linia programu źródłowego zaopatrzona jest w swój numer oraz heksadecymalą wartość licznika lokacji. Dla linii zawierających deklaracje licznik określa względne adresy zmiennych w segmencie stosu związanym z instancją podprogramu, zaś dla linii zawierających instrukcje - względne adresy instrukcji w kodzie modułu wynikowego.

Ustawiając lub kasując odpowiednią opcję przy wywołaniu kompilatora użytkownik otrzymuje pełną (wszystkie linie programu) lub skróconą (tylko linie zawierające błedy) wersję listingu. Ewentualne błędy syntaktyczne są sygnalizowane numerami, których znaczenie jest wyjaśnione na końcu listingu.

W makrozleceniu PASCAL wprowadzono dodatkowe udogodnienie związane z listowaniem błędów konsolidacji. Błędy te są wykrywane przez program LINK, a nie PASCOM/PASEN, i mogą się pojawić tylko wtedy, gdy sama kompilacja zakończyła się

pomyślnie (inaczej do konsolidacji w ogóle nie dojdzie). Podstawowym rodzajem błędu jest nieznalezienie w żadnej bibliotece pewnych podprogramów zewnętrznych, które zostały zadeklarowane w programie źródłowym. Makrozlecenie PASCAL wyśle informacje o błędach fazy konsolidacji na ten sam zbiór (domyślnie MON), o którym była mowa w opisie listingu programu.

W przypadku użycia opcji ?MAP konsolidator dopisuje do pliku listowania pełną mapę ładowania programu, niezależnie od zaistnienia ewentualnych błędów.

6. Wykonanie programu.

Skompilowany program wywołuje się zgodnie z regułami CROOKowskiego job-controla. Makrozlecenie PASCAL nadaje standardowo zbiorowi wynikowemu atrybut 064, co pozwala na parametryzację wywołania programu zgodnie z podstawowa konwencją CROOKa. W miarę upodobań atrybut ten można "ręcznie" zmienić na 054. Zwracamy uwagę na cyfrę '4' atrybutu, która informuje system o specyficznym sposobie ładowania programu (według binariów produkowanych przez LINK, a nie według absolutnego obrazu pamięci charakterystycznego dla wcześniejszych wersji CROOKa).

Jezeli w trakcie wykonania programu nie w

Jeżeli w trakcie wykonania programu nie wystąpią błędy, to po jego zakończeniu sterowanie powróci do job-controla. W przypadku nienormalnego zakończenia programu wypisana zostaje informacja o przyczynie przerwania pracy. Dalsze postępowanie zależy od ustawienia odpowiedniej opcji kompilacyjnej (por. 5.1, 5.2): sterowanie może przejść albo do job-controla, albo do interakcyjnego programu PMD (postmortem dump).

Używając komend PMD można obejrzeć wartości zmiennych lokalnych wszystkich niezakończonych procedur i funkcji (łańcuch dynamiczny), wliczając w to program główny. PMD używa strumieni INT - wejście i MON - wyjście. Po wypisaniu diagnostyki błędu PMD ma dostęp do ostatnio wykonywanej jednostki programowej. Przy przechodzeniu do kolejnych podprogramów znajdujacych się w łańcuchu dynamicznym wypisywana zostaje nazwa i względny adres zatrzymania podprogramu. Kończąc pracę z PMD użytkownik może powrócić do wykonywania programu lub zakończyć jego działanie.

6.1. Lista komend programu PMD.

R - Wznowienie działania jednostki programowej aktualnie dostępnej w łańcuchu dynamicznym (nie musi to być

akurat ta jednostka, w której nastąpiło pierwotne przerwanie pracy).

Q - Wyjście spod kontroli PMD i zakończenie wykonywania programu.

C - Ponowne wypisanie informacji o bieżącej jednostce programowej (przydatne, gdy w trakcie oglądania stosu zdążyliśmy zapomnieć, gdzie jesteśmy).

B - Przejście do poprzedniego elementu w łańcuchu dynamicznym. Wypisana zostaje nazwa i adres, pod którym nastąpiło zatrzymanie danej jednostki programowej.

A - Wypisanie całego łańcucha dynamicznego programu.

I - Powrót PMD do stanu początkowego, to znaczy na koniec łańcucha dynamicznego.

D a,l - Wypisanie zawartości pamięci (segmentu danych) aktualnie dostępnej jednostki programowej. Parametr "a" oznacza względny adres początku drukowanego obszaru stosu, zaś "l" jego długość, przy czym 0<l<=160. Pierwszy parametr podawany jest heksadecymalnie, drugi dziesiętnie. Zawartość pamieci wypisywana jest po 8 słów w linii. Na poczatku każdej linii podawany jest także względny i absolutny adres pierwszego słowa wypisanego w linii. Dla ułatwienia pracy wprowadzono domyślną postać komendy D - kolejne 8 słów stosu można obejrzeć wysyłając pustą linię.

X a,l - Wydruk dowolnego fragmentu pamięci programu. Znaczenie parametrów analogiczne jak w komendzie "D", przy czym pierwszy parametr jest absolutnym adresem początku wypisywanego obszaru.

F f,l - Zmiana formatu wypisywania, określonego parametrem f: h - oznacza wypisywanie heksadecymalne, d - dziesiętne, c - znakowe. Parametr l oznacza liczbę wypisywanych słów pamięci przy pominięciu analogicznego parametru w komendach D i X. Domyślnymi wartościami parametrów f,l jest h,8.

6.2. Lista błędów wykonania.

Numery po lewej stronie oznaczają kody błędów zwracane przez program jako tzw. wartość zewnętrzna (w rejestrze R4) w przypadku jego nienormalnego zakończenia; przy prawidłowym zakończeniu wartość zewnętrzna jest równa 0. Cecha ta może mieć znaczenie przy projektowaniu makrozleceń job-controla zawierających programy Pascalowe. Zwracana wartość zewnętrzna

jest taka sama niezależnie od tego, czy przerwanie programu nastąpiło natychmiast, czy też użytkownik wystosował dyrektywę Q po dokonaniu oceny sytuacji za pomocą PMD.

1	INDEX OUT OF RANGE	7	BAD PARAMETER VALUE
2	CASE OUT OF RANGE	8	SYSTEM ERROR nnn AT #nnnn
3	SCALAR OUT OF RANGE	9	TRIED TO READ PAST EOF
4	BAD POINTER VALUE	10	TRIED TO WRITE WHILE NOT EOF
5	BAD SIZE PARAMETER	11	NUMBER FORMAT ERROR
6	RUNTIME STACK OVERFLOW	12	HALT
		13	FILE BUFFER NOT IN BYTE AREA

ad 1 - Wyjście poza zakres przy odwołaniu do tablicy.

ad 2 - Selektor instrukcji CASE nie należy do zdefiniowanej listy. Nie dotyczy to, ze zrozumiałych względów, instrukcji CASE z selektorem domyślnym (ELSE).

ad 3 - Wyjście poza zakres przy podstawieniu na zmienną typu skalarnego. Błedy 1,2,3 są wykrywane tylko wtedy, gdy program był kompilowany z odpowiednią opcją (por. 5.1, 5.2).

ad 4 - Wskaźnik wskazuje na obiekt nie leżący w zakresie heapu.

ad 5 - Niepoprawny opis pola przy wypisywaniu napisu, liczby lub znaku.

ad 6 - Przepełnienie pamięci roboczej programu. Running-system Pascala automatycznie żąda nowych ramek pamięci po 4K słów w miarę rośnięcia stosu (w górę, zaczynając bezpośrednio za kodem programu) oraz heapu (w dół, zaczynając od adresu #FFFF). Przecięcie tych dwóch struktur danych oznacza zarazem przepełnienie maksymalnej przestrzeni adresowej 64K.

ad 7 - Niepoprawna wartość parametru podprogramu standardowego.

ad 8 - Wystąpił błąd systemowy o podanym kodzie oktalnym; wyjaśnienia tego błędu należy szukać w podręczniku CROOKa. W niektórych przypadkach nie jest możliwe prawidłowe korzystanie z PMD, dlatego też na wszelki wypadek podawany jest absolutny adres wystąpienia błędu.

ad 9 - Próba czytania po napotkaniu końca pliku, lub czytanie po operacji pisania.

ad 10- Próba pisania w środku pliku.

ad 11- Zły format liczby na wejściu.

ad 12 - Wywołana została biblioteczna procedura HALT.

ad 13 - Zmienna buforowa pliku znajduje się na stosie w takim miejscu, że jego absolutny adres jest większy niż 32K.

7. Tworzenie podprogramów zewnętrznych.

Ostatni rozdział podręcznika uzupełnia treść punktu 4.3 i jest przeznaczony dla bardziej zaawansowanych użytkowników, którzy chcieliby nie tylko korzystać z gotowych (bibliotek) podprogramów zewnętrznych, ale sami takie podprogramy i biblioteki tworzyć.

7.1 Podprogramy zewnętrzne w Pascalu.

Zanim będziemy mogli korzystać w swoim programie z podprogramów typu EXTERNAL, należy włączyć je do odpowiedniej biblioteki. Jeśli moduły biblioteczne są też napisane w Pascalu, to zadanie jest o tyle ułatwione, że ich sprzęg programowy (sposób przekazywania parametrów, wyników i sterowania) jest z natury rzeczy dostosowany do konwencji Pascala. W sumie problem przygotowania modułów bibliotecznych sprowadza się w tym wypadku do problemu niezależnej kompilacji podprogramów.

Do zadania tego przystępujemy w sposób następujący. Zbiór podprogramów przeznaczonych do umieszczenia w bibliotece obudowujemy "szczątkowym" programem głównym, np.

```
PROGRAM PUSTY;
  PROCEDURE nazwa1(..) .. END;
  FUNC          BEGIN END.
```

a następnie przetwarzamy makrozleceniem PASCAL z opcją ?ASM, względnie dokonujemy "ręcznej" kompilacji i asemblacji (ale nie konsolidacji). W rezultacie otrzymujemy zbiór zawierający relokowalne moduły żądanych podprogramów, które możemy następnie włączyć za pomocą programu LIAR do dowolnie wybranej biblioteki.

Zbiór wynikowy zawiera też - niestety - poprawnie zbudowany, ale do niczego nie potrzebny moduł programu głównego, którego oczywiście nie należy nigdzie dołączać.

Należy przyjąć jako zasadę, że procedury biblioteczne nie

korzystają z żadnych obiektów poza własnymi parametrami i zmiennymi lokalnymi. Teoretycznie rzecz biorąc, możliwe jest zadeklarowanie takiego środowiska w treści programu PUSTY, że procedura zewnętrzna umieszczona w programie docelowym odwoływałaby się do zmiennych nielokalnych o właściwych adresach na właściwych poziomach syntaktycznych. Przedsięwzięć tego rodzaju nie polecamy jednak nawet największym ekspertom - nie mówiąc już o tym, że procedurę taką trudno uznać za biblioteczną (nie pasuje do wszystkich programów).

7.2 Podprogramy zewnętrzne w asemblerze.

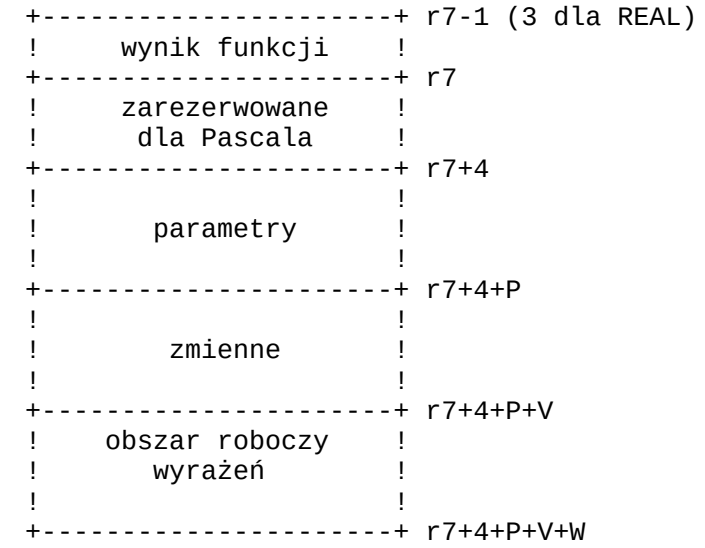
W wielu sytuacjach zachodzi potrzeba dołączenia do programu Pascalowego podprogramu napisanego w asemblerze (GASSie). W takim przypadku konieczne jest dostosowanie sposobu przekazywania parametrów, wyników i sterowania do standardu obowiązującego w Pascalu 400. Szkielet podprogramu typu EXTERNAL przedstawia się następująco:

```
          .unit      dowol
          .exp       dowol
*
* to dla PMD
          .text      'dowol  '   ; 8 pierwszych znaków nazwy
          .data      4+P+V       ; długość segmentu stosu (bez
"W")
* to dla running-systemu
          .data      0           ; poziom syntaktyczny
*
* właściwe ciało podprogramu
dowol
* przypuśćmy, że pobieramy pierwszy parametr do r1
           set   r1, (r7+4)
*          ...
* przypuśćmy, że zapisujemy wynik funkcji otrzymany w r6
           store r6, (r7-1)
* powrót przez runtime'ową procedurę .ret
          .imp      .ret
           buc      .ret
          .end
```

Poziom syntaktyczny określa głębokość zagnieżdżenia deklaracji podprogramu w tekście źródłowym: 1 dla programu głównego, 2 dla podprogramów w nim zadeklarowanych, 3 dla podprogramów zadeklarowanych w tych podprogramach, itd. Informacja ta jest potrzebna wyłącznie przy dostępie do nielokalnych zmiennych, dlatego też dla procedur bibliotecznych zarezerwowano umowny poziom 0.

Segment stosu związany z instancją podprogramu ma

następującą strukturę:



Rejestr R7 przechowuje zawsze adres bazowy segmentu stosu i jego wartość nie powinna być zmieniona przy wyjściu z procedury. Pierwsze 4 słowa segmentu zarezerwowane są dla running-systemu Pascala (poziom syntaktyczny, modyfikacja display'a, dowiązanie dynamiczne, adres powrotny) i najlepiej też ich w ogóle nie dotykać.

Dalej zaczyna się pole parametrów, które umieszczone są na stosie zgodnie z ich kolejnością w nagłówku podprogramu. Parametry przekazywane przez zmienną (VAR) oraz parametry proceduralne/funkcyjne zajmują jedno słowo (adres zmiennej lub punktu wejścia do podprogramu). Parametry przekazywane przez wartość zajmują zawsze liczbę słów zgodną z ich rozmiarem np.: REAL - 3, CHAR - 1, zmienne strukturalne - całkowity rozmiar. Wartość funkcji zwracana jest na stosie pod adresem R7-3 dla typu REAL, zaś dla pozostałych typów pod adresem R7-1.

Długość segmentu stosu dla podprogramu napisanego w asemblerze jest prawie zawsze równa 4+P, ponieważ nie ma powodu, aby jego własne zmienne robocze musiały być przechowywane akurat na stosie (chyba, że nasze ambicje sięgają asemblerowych programów rekursywnych).

7.3 Podprogramy zewnętrzne w innych językach

W ostateczności Pascal może korzystać z podprogramów zewnętrznych napisanych w (niemal) dowolnym języku programowania, względnie należących do standardowej biblioteki innego języka. Należy zacząć od dokładnego zapoznania się z

konwencjami wywołania/powrotu obowiązującymi w Pascalu i w języku X, a następnie napisać w GASSie zespół prostych procedur sprzęgających.

Dobrymi kandydatami na język X wydają się Fortran i C, pod warukiem jednak, że ich kompilatory dostępne pod CROOKiem generowałyby kod akceptowalny przez GASS lub bezpośrednio przez LINK.